

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application

Applicant(s): Arun Kwangil Iyengar
Docket No.: YOR920010663US1
Serial No.: 10/629,284
Filing Date: July 29, 2003
Group: 2188
Examiner: Duc T. Doan

Title: Methods and Systems for Managing Persistent
Storage of Small Data Objects

AMENDED APPEAL BRIEF

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

This Amended Appeal Brief is being filed in response to the Notification of Non-Compliant Appeal Brief dated January 18, 2007.

Applicants (hereinafter referred to as "Appellants") hereby appeal the final rejection of claims 1-28 of the above-identified application.

REAL PARTY IN INTEREST

The present application is assigned to International Business Machines Corporation, as evidenced by an assignment recorded July 29, 2003 in the U.S. Patent and Trademark Office at Reel 14329, Frame 799. The assignee, International Business Machines Corporation, is the real party in interest.

RELATED APPEALS AND INTERFERENCES

Applicants are not aware of any related appeals or interferences.

STATUS OF CLAIMS

Claims 1-28 stand finally rejected under 35 U.S.C. §103(a). Claims 1-28 are appealed.

STATUS OF AMENDMENTS

There has been no amendment filed subsequent to the final rejection.

SUMMARY OF CLAIMED SUBJECT MATTER

As the background section of the present application explains, a key problem in data storage management is how to efficiently manage objects which are relatively small but need to be stored persistently, such as in disk storage rather than in main memory only. One of the key issues for storing such objects is that the minimum unit of transfer, known as a sector, is often much larger than the minimum object size. By way of example, a sector size may be more than a factor of two larger than some objects. This means that the disk storage system will transfer sectors between disk and main memory. When an object is much smaller than a sector, the disk storage system does not have a method for transferring just the object. The disk storage system has to transfer the entire sector (or sectors if the object is on a sector boundary) containing the object in order to access just the object (Specification, page 1, lines 8-20).

Independent claim 1 recites a method of managing storage of objects of sizes smaller than a storage transfer unit in a computer system, comprising the steps of: maintaining a plurality of storage transfer units in a first storage medium organized by a quantity of free space in a storage transfer unit; maintaining in a second storage medium a cache comprising a copy of at least one of said plurality of storage transfer units; in response to a request to store an object of a size less than a storage transfer unit: searching for a cached storage transfer unit with sufficient free space to store the object; if no such cached storage transfer unit can be found, identifying an uncached storage transfer unit with sufficient free space to store the object and storing a copy of the identified storage transfer unit in the cache; and storing the object in the identified storage transfer unit by modifying at least one data structure in the cache and subsequently writing a cached copy of the storage transfer

unit to the first storage medium. The present specification provides an illustrative embodiment of the elements of claim 1 at page 7, line 17, through page 9, line 21.

More particularly, FIG. 3, a flow diagram, illustrates a methodology for storing a new persistent object, according to an embodiment of the invention. That is, FIG. 3 shows an illustrative method for satisfying an allocation request for a new object smaller than a sector size. This methodology is performed under control of SODM 104. In step 310, it is determined if the request is for customized allocation. If so, the application (e.g., one or more computer programs utilizing the disk storage system to execute their functions) handles the request in step 320. The application may invoke application programming interface (API) functions on SODM 104 to perform customized allocation. For example, an application might indicate that several objects should be stored on the same sector for better locality of reference. This would reduce the number of disk accesses for objects which are likely to be accessed in close temporal locality to each other. The application could also specify that specific objects be mapped to specific sectors. This could also improve performance by achieving better locality. If the request is not for customized allocation, it is determined in step 330 whether there is space for the object in a cached sector, i.e., a sector which has been copied to the cache with sufficient free space. SODM 104 maintains an ordered data structure containing the number of free bytes in cached sectors, fb. The data structure fb can be implemented in several ways, such as a balanced tree which is efficient or a list which is not as efficient. If there are multiple cached sectors with sufficient free space, several methods may be used to select an appropriate sector. One method is to select a sector with the least free space sufficient to accommodate the new object. This approach is known as a best fit approach. If a sector is found with sufficient free space, processing continues to step 340.

Referring now to FIG. 4, an example of how a cached sector might be represented in memory is shown. In this figure there are 8 (although the invention is not limited thereto) pointers to buffers. Buffers b1, b3, b4, b6, and b7 contain objects. Pointers 2, 5, and 8 are null because the objects they previously pointed to have been deleted. In this particular example, a buffer for the new object could be created (or an existing buffer could be reused) and stored in position 5. Position 5 would be removed from the deleted object list. The total allocated bytes would be reduced by the space taken

up by the new object, and fb would be updated. If, on the other hand, the deleted object list were empty and pointers 2, 5, and 8 pointed to buffers containing objects, an additional pointer 9 would be created to point to the buffer for the new object.

Returning now to FIG. 3, if a cached sector with sufficient free space is not found in step 330, processing continues to step 350. In step 350, SODM 104 attempts to locate an uncached sector with sufficient space for the new object. SODM 104 does so by examining at least one free list (free lists are illustrated in FIG. 2). Free lists are maintained on disk for persistence. They may also be cached in main memory for better performance. If they are cached in main memory, step 350 can generally execute more quickly. If there are multiple cached sectors with sufficient free space, several methods may be used to select an appropriate sector. One method is to select a sector on a free list with the most free space. Since free lists are organized by size, this can be done by examining free lists starting from the one storing the largest blocks and moving downwards until either a nonempty list is found or a free list is reached with insufficient space to store the new object (in which case a tail is examined to obtain a sector with enough free space). If a sector with sufficient space is identified in step 350, the sector is cached in main memory in step 360. The method for doing so is similar to that in step 506 of FIG. 5. Space is then allocated for the new object in the manner described for step 340. Since the newly cached sector will not have anything on its deleted object list, the new object will be stored as the last buffer associated with the sector. If an appropriate sector is not identified in step 350, an empty sector is allocated from a tail in step 370. The tail pointer corresponding to the allocation request is then modified to point to the next unallocated sector. Performing allocations from a tail too frequently can fragment disks and require more disk space for storage. By only resorting to tail allocations as a last step, the approach in FIG. 3 minimizes fragmentation. (Specification, page 7, lines 17 through page 9, line 21).

Independent claim 16 recites a method of maintaining a plurality of objects in a storage transfer unit, comprising the steps of: identifying an object position in the storage transfer unit by an object offset in the storage transfer unit; in response to a request to one of access and update a storage transfer unit, copying the storage transfer unit so that different objects are copied into different buffers; performing at least one update to at least one object in the storage transfer unit by

modifying at least one buffer; and after the at least one update has occurred, updating the storage transfer unit from the at least one buffer. The present specification provides an illustrative embodiment of the elements of claim 16 at page 9, line 22 through page 11, line 13.

More particularly, FIG. 5, a flow diagram, illustrates a methodology for accessing or updating an object, according to an embodiment of the invention. This methodology is performed under control of SODM 104. In step 502, a request to access/update an object is received. An object may be identified by its sector identification (ID) and object offset within its sector. For example, if an object is the 10th object in sector x, then its object offset is 10. In step 504, it is determined if the sector is already cached. If so, then processing continues to step 508, wherein the access or update is performed. As an example of an access, in order to access the 3rd object of the cached sector depicted in FIG. 4, the contents of buffer b3 would be returned. As an example of an update, in order to update the 6th object of the cached sector depicted in FIG. 4 with a new object which is not big enough to overflow the sector, buffer b6 is replaced with a buffer containing the new object. In some cases, the old buffer can be reused and/or resized to contain the new object. As another example of an update, suppose that the 6th object of the cached sector depicted in FIG. 4 is updated to a size which would overflow the sector. FIG. 6 is a flow diagram illustrating a methodology for handling a sector overflow, according to an embodiment of the invention. This methodology is performed under control of SODM 104. If it is determined in step 602 that the growth of an object overflows the sector, then a new sector with sufficient space is located for the updated object, and the updated object is stored in the new sector in step 604. This process would be similar to that previously mentioned for FIG. 3, starting at step 330. The old location for the object may then be replaced by a forwarding pointer of the format depicted in FIG. 7. The negative tag indicates a forwarding pointer as opposed to object data as depicted in FIG. 8. The forwarding pointer contains the new location for the object. That way, the object can be located from its old location.

Returning to FIG. 5, if it is determined in step 504 that the sector is not cached, the sector is cached in step 506. FIG. 9 shows a method by which a sector on disk might be arranged. A header might contain information such as the number of objects in the sector and the number of free bytes. Other information might be contained in a header as well. After the header, data corresponding to

the objects would be stored. An object would be of the form depicted in FIG. 8, while a forwarding pointer would be of the form depicted in FIG. 7. Size headers for objects allow SODM 104 to determine when one object ends and another begins. Forwarding pointers are of a fixed size known by SODM 104. In some situations, it may not be necessary to cache all of a sector. In other situations, it may be acceptable to cache an inexact copy of a sector. SODM 104 reads in the sector which might be of the format depicted in FIGs. 7, 8 and 9 and caches the sector in main memory. The cached main memory format may be similar to the one depicted in FIG. 4. SODM 104 might also maintain other information about cached sectors, such as information correlated with how frequently the sector is accessed. This information could be used to determine which cached object to throw out when the cache overflows (i.e., a cache replacement policy). One such policy is LRU, or least recently used. Under this policy, the cached object accessed the least recently is replaced. Other replacement policies could be used as well. (Specification, page 9, line 22 through page 11, line 13).

Independent claim 25 is an apparatus claim having similar elements as the above-described claim 1. The apparatus comprises a processor and memory arrangement. FIG. 12 shows an illustrative hardware implementation of a computing system in accordance with which one or more components/methodologies of a small object disk management system may be implemented. As shown, the computer system may be implemented in accordance with a processor 1202, a memory 1204, I/O devices 1206, and a network interface 1208, coupled via a computer bus 1210 or alternate connection arrangement. (Specification, page 12, line 23 through page 13, line 12).

Independent claim 26 is an apparatus claim having similar elements to the above-described claim 16. The apparatus comprises a processor and memory arrangement. FIG. 12 shows an illustrative hardware implementation of a computing system in accordance with which one or more components/methodologies of a small object disk management system may be implemented. As shown, the computer system may be implemented in accordance with a processor 1202, a memory 1204, I/O devices 1206, and a network interface 1208, coupled via a computer bus 1210 or alternate connection arrangement. (Specification, page 12, line 23 through page 13, line 12).

Independent claim 27 is an article of manufacture claim having similar elements as the above-described claim 1. The article of manufacture comprises a machine readable medium. The present specification provides an illustrative embodiment of the elements of claim 27 at page 14, lines 3-7. More particularly, as explained therein, software components including instructions or code for performing the methodologies described herein may be stored in one or more of the associated memory devices (e.g., ROM, fixed or removable memory) and, when ready to be utilized, loaded in part or in whole (e.g., into RAM) and executed by a CPU. (Specification, page 14, lines 3-7).

Independent claim 28 is an article of manufacture claim having similar elements as the above-described claim 16. The article of manufacture comprises a machine readable medium. The present specification provides an illustrative embodiment of the elements of claim 27 at page 14, lines 3-7. More particularly, as explained therein, software components including instructions or code for performing the methodologies described herein may be stored in one or more of the associated memory devices (e.g., ROM, fixed or removable memory) and, when ready to be utilized, loaded in part or in whole (e.g., into RAM) and executed by a CPU. (Specification, page 14, lines 3-7).

GROUND OF REJECTION TO BE REVIEWED ON APPEAL

(I) Whether claims 1-3, 9, 11, 12, 14-16 and 19-28 are unpatentable under 35 U.S.C. §103(a) over U.S. Publication No. 2002/0032691 filed in the name of Rabii et al. (hereinafter “Rabii”) in view of U.S. Patent No. 6,915,307 issued to Mattis et al. (hereinafter “Mattis”).

(II) Whether claims 4-8 and 10 are unpatentable under 35 U.S.C. §103(a) over Rabii in view of Mattis in further in view of U.S. Patent No. 6,804,761 issued to Chen et al. (hereinafter “Chen”).

(III) Whether claims 13 and 17 are unpatentable under 35 U.S.C. §103(a) over Rabii in view of Mattis in further in view of U.S. Patent No. 5,802,599 issued to Carbrera et al. (hereinafter “Cabrera”).

(IV) Whether claim 18 is unpatentable under 35 U.S.C. §103(a) over Rabii in view of Mattis in further in view of Cabrera in still further view of U.S. Publication No. 2004/0172507 filed in the name of Garthwaite (hereinafter “Garthwaite”).

ARGUMENT

Appellants incorporate by reference herein the disclosure of their previous response filed in the present application, namely, the response dated November 30, 2005.

(I) Whether claims 1-3, 9, 11, 12, 14-16 and 19-28 are unpatentable under 35 U.S.C. §103(a) over U.S. Publication No. 2002/0032691 filed in the name of Rabii et al. (hereinafter “Rabii”) in view of U.S. Patent No. 6,915,307 issued to Mattis et al. (hereinafter “Mattis”).

Regarding the §103(a) rejections, Applicant asserts that the various references, alone or in combination, fail to teach or suggest all of the limitations of the claims 1-28, as will be explained below. Furthermore, with regard to the combinations of the various references, Applicant asserts that such combinations are improper, as will be explained below.

The Examiner cites Rabii in combination with Mattis in rejecting independent claims 1, 16 and 25-28. More particularly, the Examiner cites portions of Rabii as disclosing certain limitations of the independent claims, and cites portions of Mattis as disclosing certain other limitations of the independent claims. Below Applicant explains how such portions of Rabii and Mattis fail to teach or suggest what the Examiner contends that they teach or suggest. While Applicant may refer from time to time to each reference alone in describing its deficiencies, it is to be understood that such arguments are intended to point out the overall deficiency of the cited combination.

The Rabii patent application is directed to methods for maintaining directory structures on disk. Rabii does not disclose how to allocate space efficiently for small objects as independent claim 1 recites. Claim 1 explicitly recites that it is a method of managing storage of “objects of sizes smaller than a storage transfer unit.”

As pointed out in the background section of the present application and repeated above, a key problem in data storage management is how to efficiently manage objects which are relatively small

but need to be stored persistently, such as in disk storage rather than in main memory only, wherein one of the key issues for storing such objects is that the minimum unit of transfer (also referred to as a storage transfer unit), such as a sector, is often much larger than the minimum object size. As explained, a sector size may be more than a factor of two larger than some objects. This means that the disk storage system will transfer sectors between disk and main memory. When an object is much smaller than a sector, the disk storage system does not have a method for transferring just the object. The disk storage system has to transfer the entire sector (or sectors if the object is on a sector boundary) containing the object in order to access just the object.

Thus, the steps of the claimed invention are directed toward managing storage of “objects of sizes smaller than a storage transfer unit.” Rabii does not introduce any notion of a storage transfer unit or objects of sizes smaller than a storage transfer unit. The final Office Action fails to address this deficiency other than to state (at page 11) that “transferring a fix [sic] amount, for example a cache line between the disk and a cache met the . . . definition.”

Regarding the contention in the final Office Action that Rabii discloses the first step of claim 1 (i.e., maintaining a plurality of storage transfer units in a first storage medium organized by a quantity of free space in a storage transfer unit), Applicant strongly disagrees. First, data object partitions of Rabii are not the same as storage transfer units of the claimed invention. There is no support given in the final Office Action or in Rabii to conclude otherwise. Secondly, Rabii does not organize such data object partitions by a quantity of free space, as the claimed invention recites with respect to storage transfer units.

Furthermore, Rabii fails to disclose the second step of claim 1 (i.e., maintaining in a second storage medium a cache comprising a copy of at least one of said plurality of storage transfer units) since there is no notion of storage transfer units maintained in the data buffers of FIG. 10 of Rabii.

Regarding the contention that Mattis discloses some steps of claim 1, Applicant strongly disagrees. Mattis discloses a method for managing an object cache, i.e., the data objects are stored directly in the cache. The claimed invention recites storing an object of a size less than a storage transfer unit in a storage transfer unit of the cache (which is maintained in the second storage

medium) and then writing a cached copy of the storage transfer unit to the first storage medium. This extra level of indirection in the claimed invention further distinguishes it from Mattis.

Still further, the combination of Rabii and Mattis is improper. These two cited references are solving completely different problems. Rabii is directed to the problems associated with maintaining directory structures on a disk, while Mattis is directed to the problems associated with managing an object cache. Thus, it is not clear how or why one would combine the two disparate references.

The Federal Circuit has stated that when patentability turns on the question of obviousness, the obviousness determination “must be based on objective evidence of record” and that “this precedent has been reinforced in myriad decisions, and cannot be dispensed with.” In re Lee, 277 F.3d 1338, 1343 (Fed. Cir. 2002). Moreover, the Federal Circuit has stated that “conclusory statements” by an examiner fail to adequately address the factual question of motivation, which is material to patentability and cannot be resolved “on subjective belief and unknown authority.” Id. at 1343-1344.

In the final Office Action at page 4, the Examiner provides the following statement to prove motivation to combine Rabii and Mattis, with emphasis supplied: “[i]t would have been obvious to one of ordinary skill in the art at the time of the invention to include the steps for storing an object as suggested by Mattis in Rabii’s system to assure the integrity of information objects . . .”

Applicant submits that this statement is based on the type of “subjective belief and unknown authority” that the Federal Circuit has indicated provides insufficient support for an obviousness rejection. More specifically, the Examiner fails to identify any objective evidence of record which supports the proposed combination. The Examiner cites Mattis at column 21, lines 1-15, which states that “the foregoing sequence of steps is ordered in a way that ensures the integrity of information objects that are written to the cache.” However, since Rabii has nothing to do with the problems associated with managing an object cache, but rather is directed to the problems associated with maintaining directory structures on a disk, it is unclear how the cited portion of Rabii provides proper motivation to combine Rabii and Mattis. The final Office Action gives no further guidance on this important point.

For at least these reasons, Applicant asserts that independent claim 1 and independent claims 25 and 27 (which contain similar limitations as independent claim 1) are patentable over the Rabii/Mattis combination. Furthermore, Applicant asserts that the claims which depend from claim 1 are patentable over the Rabii/Mattis combination not only for the reasons given above with respect to claim 1, but also because such dependent claims recite patentable subject matter in their own right, as will be set out below.

Regarding independent claim 16, Applicant asserts that neither Rabii nor Mattis, alone or in combination, teach or suggest all of the limitations of the claimed invention. Applicant notes that the final Office Action corrects the wrong reference cited in the non-final Office Action (Rabii when the non-final Office Action apparently meant Mattis) in the one paragraph argument against claim 16. However, the rationale for rejecting the claim in the final Office Action is still deficient since various limitations in claim 16 are not disclosed by the cited combination. By way of example only, there is nothing the same or analogous in the cited combination to the claimed features of “storage transfer units” and “object offsets,” and how they relate to one another. Also, Rabii is silent to copying a storage transfer unit, in response to a request to one of access and update a storage transfer unit, so that different objects are copied into different buffers.

Further, the Rabii/Mattis combination is improper, as explained above.

For at least these reasons, Applicant asserts that independent claim 16 and independent claims 26 and 28 (which include similar limitations as independent claim 16) are patentable over the Rabii/Mattis combination. Furthermore, Applicant asserts that the claims which depend from claim 16 are patentable over the Rabii/Mattis combination not only for the reasons given above with respect to claim 16, but also because such dependent claims recite patentable subject matter in their own right, as will be set out below.

Regarding claims 2 and 3, whether or not Rabii refers to disk storage and main memory is not relevant, since Rabii fails to refer to specific steps for managing storage of objects of sizes smaller than a storage transfer unit in the context of disk storage and main memory, as in the claimed invention.

Regarding claims 9 and 11, any disclosed data partitions or pointers in Rabii fail to function in the specific ways that claims 9 and 11 respectively recite.

Regarding claim 12, it is not clear how the fact that Rabii describes that “objects remain in cache until there is a shortage of in-memory-objects or data buffers . . . [wherein] the object is purged from cache,” teaches or suggests that a cached copy of a storage transfer unit (from the second storage medium) is written to the first storage medium in response to at least one of the criterion recited in claim 12.

Regarding claim 14, it is unclear how a “large write back” to the disk in Rabii during a system shutdown supports a rejection of the claimed features of purging the cached copy from the cache as a result of at least one of a cache replacement policy and the computer system being about to go down.

Regarding claims 19 and 20-24, since Rabii does not teach or suggest the claimed features of “storage transfer units” and “object offsets” or the step of copying a storage transfer unit, in response to a request to one of access and update a storage transfer unit, so that different objects are copied into different buffers, it is not clear how the features specifically defined in claims 19 and 20-24 could be taught or suggested by the cited portions of Rabii.

(II) Whether claims 4-8 and 10 are unpatentable under 35 U.S.C. §103(a) over Rabii in view of Mattis in further in view of U.S. Patent No. 6,804,761 issued to Chen et al. (hereinafter “Chen”).

Applicant asserts that the claims 4-8 and 10, which depend from claim 1, are patentable over the Rabii/Mattis/Chen combination not only for the reasons given above with respect to claim 1, but also because such dependent claims recite patentable subject matter in their own right.

Whether or not Chen describes a “chunk manager capable of searching for the next highest size standard memory blocks to fit an application request,” or “searching through a range of blocks with different sizes . . . thereby reducing fragmentation,” or “memory pools with lists of standard blocks,” is not relevant to the rejection of the claimed features in claims 4-8 and 10 due at least to the failure of any of the references in the cited combination to recognize an object of a size less than a storage transfer unit.

It is also asserted that the motivation set forth by the Examiner to combine Chen with Rabii and Mattis is insufficient under In re Lee decision (cited above).

(III) Whether claims 13 and 17 are unpatentable under 35 U.S.C. §103(a) over Rabii in view of Mattis in further in view of U.S. Patent No. 5,802,599 issued to Cabrera et al. (hereinafter “Cabrera”).

Applicant asserts that the claims 13 and 17, which depend from claims 1 and 16 respectively, are patentable over the Rabii/Mattis/Cabrera combination not only for the reasons given above with respect to claims 1 and 16, but also because such dependent claims recite patentable subject matter in their own right.

Cabrera does not disclose the notion of an “object offset.” The portion of Cabrera disclosing reuse of a memory buffer fails to teach or suggest such feature.

It is also asserted that the motivation set forth by the Examiner to combine Cabrera with Rabii and Mattis is insufficient under In re Lee decision (cited above).

(IV) Whether claim 18 is unpatentable under 35 U.S.C. §103(a) over Rabii in view of Mattis in further in view of Cabrera in still further view of U.S. Publication No. 2004/0172507 filed in the name of Garthwaite (hereinafter “Garthwaite”).

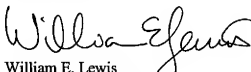
Applicant asserts that the claim 18, which depends from claim 16, is patentable over the Rabii/Mattis/Cabrera/Garthwaite combination not only for the reasons given above with respect to claim 1, but also because such dependent claim recites patentable subject matter in their own right.

First, Cabrera does not perform the steps that the Examiner suggests it does on page 10 of the final Office Action. Further, Garthwaite does not disclose the notion of an “storing a forwarding pointer in the previous storage transfer unit.”

It is also asserted that the motivation set forth by the Examiner to combine Garthwaite with Rabii, Mattis and Cabrera is insufficient under In re Lee decision (cited above).

In view of the above, Applicants believe that claims 1-28 are in condition for allowance, and respectfully request withdrawal of the various §103(a) rejections.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "William E. Lewis". The signature is fluid and cursive, with the first name "William" being more prominent.

Date: February 15, 2007

William E. Lewis
Attorney for Applicant(s)
Reg. No. 39,274
Ryan, Mason & Lewis, LLP
90 Forest Avenue
Locust Valley, NY 11560
(516) 759-2946

APPENDIX

1. A method of managing storage of objects of sizes smaller than a storage transfer unit in a computer system, comprising the steps of:

maintaining a plurality of storage transfer units in a first storage medium organized by a quantity of free space in a storage transfer unit;

maintaining in a second storage medium a cache comprising a copy of at least one of said plurality of storage transfer units;

in response to a request to store an object of a size less than a storage transfer unit:

searching for a cached storage transfer unit with sufficient free space to store the object;

if no such cached storage transfer unit can be found, identifying an uncached storage transfer unit with sufficient free space to store the object and storing a copy of the identified storage transfer unit in the cache; and

storing the object in the identified storage transfer unit by modifying at least one data structure in the cache and subsequently writing a cached copy of the storage transfer unit to the first storage medium.

2. The method of claim 1, wherein the first storage medium comprises disk storage.

3. The method of claim 1, wherein the second storage medium comprises main memory.

4. The method of claim 1, wherein the step of searching for a cached storage transfer unit further comprises identifying a cached storage transfer unit with sufficient free space to store the object giving preference to such cached storage transfer units with less free space.

5. The method of claim 4, wherein the step of searching for a cached storage transfer unit further comprises identifying a cached storage transfer unit with a least amount of free space sufficient to store the object.

6. The method of claim 1, wherein the step of identifying an uncached storage transfer unit further comprises identifying an uncached storage transfer unit with sufficient free space giving preference to storage transfer units which minimize fragmentation.

7. The method of claim 1, wherein the step of identifying an uncached storage transfer unit further comprises giving preference to storage transfer units with more free space.

8. The method of claim 7, wherein the step of identifying an uncached storage transfer unit further comprises identifying a storage transfer unit with a most free space.

9. The method of claim 1, further comprising the steps of:
maintaining at least one list of storage transfer units;
maintaining at least one tail pointer to a plurality of contiguous unallocated storage transfer units;

wherein the step of identifying an uncached storage transfer unit further comprises the steps of searching for an uncached storage transfer unit on the at least one list with sufficient space, and if such an uncached storage transfer unit can not be located, identifying an unallocated storage transfer unit from the at least one tail pointer.

10. The method of claim 1, further comprising the step of maintaining a plurality of lists of storage transfer units organized by a quantity of free space in a storage transfer unit.

11. The method of claim 1, further comprising the step of maintaining at least one tail pointer to a plurality of contiguous unallocated storage transfer units.

12. The method of claim 1, wherein a cached copy of a storage transfer unit is written to the first storage medium in response to at least one of: (i) an object in the storage transfer unit being updated; (ii) a number of changed bytes in the storage transfer unit exceeding a threshold; (iii) a

number of changed objects in the storage transfer unit exceeding a threshold; and (iv) the cached copy being about to be purged from the cache.

13. The method of claim 1, wherein an application program writes at least one storage transfer unit to disk in a transactional manner.

14. The method of claim 12, wherein the cached copy is about to be purged from the cache as a result of at least one of a cache replacement policy and the computer system being about to go down.

15. The method of claim 1, wherein a storage transfer unit comprises a sector.

16. A method of maintaining a plurality of objects in a storage transfer unit, comprising the steps of:

identifying an object position in the storage transfer unit by an object offset in the storage transfer unit;

in response to a request to one of access and update a storage transfer unit, copying the storage transfer unit so that different objects are copied into different buffers;

performing at least one update to at least one object in the storage transfer unit by modifying at least one buffer; and

after the at least one update has occurred, updating the storage transfer unit from the at least one buffer.

17. The method of claim 16, further comprising the steps of:

when an object which does not have a highest offset is deleted, adding the offset to a list; satisfying an allocation request by using an offset from the list; and

if an offset is not reused by the time the storage transfer unit is updated from the at least one buffer, storing a placeholder on the storage transfer unit indicating the object has been deleted.

18. The method of claim 16, further comprising the step of:

in response to an object update which would cause a storage transfer unit to overflow, moving the object to a new storage transfer unit and storing a forwarding pointer in the previous storage transfer unit.

19. The method of claim 16, wherein the storage transfer unit is maintained on disk and the at least one buffer is maintained in main memory.

20. The method of claim 16, wherein the step of updating the storage transfer unit from the at least one buffer further comprises copying a plurality of objects from buffers to the storage transfer unit in a contiguous area so that free space in the storage transfer unit is contiguous.

21. The method of claim 16, wherein the storage transfer unit comprises a sector.

22. The method of claim 16, further comprising the step of maintaining a number of free bytes in the storage transfer unit.

23. The method of claim 22, wherein the step of performing at least one update further comprises using the number of free bytes in the storage transfer unit to prevent overflow.

24. The method of claim 16, wherein the copy of at least one of the plurality of storage transfer units included in the cache is one of a partial copy and an inexact copy.

25. Apparatus for managing storage of objects of sizes smaller than a storage transfer unit in a computer system, comprising:

at least one processor operative to: (i) maintain a plurality of storage transfer units in a first storage medium organized by a quantity of free space in a storage transfer unit; (ii) maintain in a second storage medium a cache comprising a copy of at least one of said plurality of storage transfer units; (iii) in response to a request to store an object of a size less than a storage transfer unit:

searching for a cached storage transfer unit with sufficient free space to store the object; if no such cached storage transfer unit can be found, identifying an uncached storage transfer unit with sufficient free space to store the object and storing a copy of the identified storage transfer unit in the cache; and storing the object in the identified storage transfer unit by modifying at least one data structure in the cache and subsequently writing a cached copy of the storage transfer unit to the first storage medium.

26. Apparatus for maintaining a plurality of objects in a storage transfer unit, comprising:
at least one processor operative to: (i) identify an object position in the storage transfer unit by an object offset in the storage transfer unit; (ii) in response to a request to one of access and update a storage transfer unit, copy the storage transfer unit so that different objects are copied into different buffers; (iii) perform at least one update to at least one object in the storage transfer unit by modifying at least one buffer; and (iv) after the at least one update has occurred, update the storage transfer unit from the at least one buffer.

27. An article of manufacture for managing storage of objects of sizes smaller than a storage transfer unit in a computer system, comprising a machine readable medium containing one or more programs which when executed implement the steps of:

maintaining a plurality of storage transfer units in a first storage medium organized by a quantity of free space in a storage transfer unit;

maintaining in a second storage medium a cache comprising a copy of at least one of said plurality of storage transfer units;

in response to a request to store an object of a size less than a storage transfer unit:

searching for a cached storage transfer unit with sufficient free space to store the object;

if no such cached storage transfer unit can be found, identifying an uncached storage transfer unit with sufficient free space to store the object and storing a copy of the identified storage transfer unit in the cache; and

storing the object in the identified storage transfer unit by modifying at least one data structure in the cache and subsequently writing a cached copy of the storage transfer unit to the first storage medium.

28. An article of manufacture for maintaining a plurality of objects in a storage transfer unit, comprising a machine readable medium containing one or more programs which when executed implement the steps of:

identifying an object position in the storage transfer unit by an object offset in the storage transfer unit;

in response to a request to one of access and update a storage transfer unit, copying the storage transfer unit so that different objects are copied into different buffers;

performing at least one update to at least one object in the storage transfer unit by modifying at least one buffer; and

after the at least one update has occurred, updating the storage transfer unit from the at least one buffer.

EVIDENCE APPENDIX

None.

RELATED PROCEEDINGS APPENDIX

None.